Last Class: Semaphores

- A semaphore S supports two atomic operations:
  - **S→Wait():** get a semaphore, wait if busy semaphore S is available.
  - **S→Signal():** release the semaphore, wake up a process if one is waiting for S.
- **Binary or Mutex Semaphore:** grants mutual exclusive access to a resource
- **Counting Semaphore:** useful for granting mutually exculsive access for a set of resources
- Semaphores are useful for mutual exclusion, progress and bounded waiting

Today: Synchronization for Readers/Writers Problem

- An object is shared among may threads, each belonging to one of two classes:
  - Readers: read data, never modify it
  - Writers: read data and modify it
- Using a single lock on the data object is overly restrictive
  - ⇒ Want *many readers* reading the object at once
  - Allow only *one writer* at any point
  - How do we control access to the object to permit this protocol?
- Correctness criteria:
  - Each read or write of the shared data must happen within a critical section.
  - Guarantee mutual exclusion for writers.
  - Allow multiple readers to execute in the critical section at once.

Readers/Writers Problem

```
class ReadWrite {
   public:
     void Read();
     void Write();
   private:
     int       readers; // counts readers
     Semaphore mutex;   // controls access to readers
     Semaphore wrt;     // controls entry to first
}                       // writer or reader
ReadWrite::ReadWrite {
   readers      = 0;
   mutex->value = 1;
   wrt->value   = 1;
}
```


Readers/Writers Problem

```
ReadWrite::Write(){
   wrt->Wait();         // any writers or readers?
   <perform write>
   wrt->Signal();       // enable others
}
ReadWrite::Read(){
   mutex->Wait();       // ensure mutual exclusion
      readers += 1;     // another reader
      if (readers == 1)
         wrt->Wait();   // block writers
   mutex->Signal();
   <perform read>
   mutex->Wait();       // ensure mutual exclusion
      readers -= 1;     // reader done
      if (readers == 0)
         wrt->Signal();// enable writers
   mutex->Signal(); }
```

Readers/Writers: Scenario 1

```
R1:                R2:               W1:
Read ()
                 Read ()
                                   Write ()
```


Readers/Writers: Scenario 2

```
R1:                R2:               W1:
                                   Write ()
Read ()
                 Read ()
```

Reader/Writers: Scenario 3

```
R1:                  R2:               W1:
Read ()
                                     Write ()
                   Read ()
```


Readers/Writers Solution: Discussion

- Implementation notes:
  1. The first reader blocks if there is a writer; any other readers who try to enter block on `mutex`.
  2. The last reader to exit signals a waiting writer.
  3. When a writer exits, if there is both a reader and writer waiting, which goes next depends on the scheduler.
  4. If a writer exits and a reader goes next, then all readers that are waiting will fall through (at least one is waiting on `wrt` and zero or more can be waiting on `mutex`).
  5. Does this solution guarantee all threads will make progress?

- Alternative desirable semantics:
  - Let a writer enter its critical section as soon as possible.

Readers/Writers Solution Favoring Writers

```
ReadWrite::Write(){
   write_mutex->Wait();  // ensure mutual exclusion
      writers += 1;      // another pending writer
      if (writers == 1)  // block readers
         read_block->Wait();
   write_mutex->Signal();
   write_block->Wait();  // ensure mutual exclusion
   <perform write>
   write_block->Signal();
   write_mutex->Wait();  // ensure mutual exclusion
      writers -= 1;      // writer done
      if (writers == 0)  // enable readers
         read_block->Signal();
   write_mutex->Signal(); }
```


Readers/Writers Solution Favoring Writers

```
ReadWrite::Read(){
   write_pending->Wait();  // ensures at most one reader will go
                           // before a pending write
   read_block->Wait();
   read_mutex->Wait();       // ensure mutual exclusion
      readers += 1;          // another reader
      if (readers == 1)      // synchronize with writers
         write_block->Wait();
   read_mutex->Signal();
   read_block->Signal();
   write_pending->Signal();
   <perform read>
   read_mutex->Wait();       // ensure mutual exclusion
      readers -= 1;          // reader done
      if (readers == 0)      // enable writers
         write_block->Signal();
   read_mutex->Signal(); }
```

Readers/Writers: Scenario 4

R1:	R2:	W1:	W2:
Read ()			
	Read ()		
		Write ()	
			Write ()

Readers/Writers: Scenario 5

R1:	R2:	W1:	W2:
		Write ()	
Read ()			
	Read ()		
			Write ()

Reader/Writers: Scenario 6

```
R1:                R2:              W1:              W2:
Read ()
                                  Write ()
                Read ()
                                                 Write ()
```


Other Synchronizations Problems: Dining Philosophers

- Five philosophers, each either eats or thinks
- Share a circular table with five chopsticks
- Thinking: do nothing
- Eating $\Rightarrow$need two chopsticks, try to pick up two closest chopsticks
  - Block if neighbor has already picked up a chopstick
- After eating, put down both chopsticks and go back to thinking
- *Reading assignment:* read OSC ch 6, pages 175-177 for a solution to this problem

Summary

- Readers/writers problem:
  - Allow multiple readers to concurrently access a data
  - Allow only one writer at a time
- Two possible solutions using semaphores
  - Favor readers
  - Favor writers
- Starvation is possible in either case!